UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Jančič

Optimizacija poizvedb v sistemu ElasticSearch

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Marko Bajec

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.
Slike so izdelane s pomočjo jezika PGF/TikZ.
Grafi so narisani s pomočjo programa ***gnuplot****.*

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Andrej Jančič,

z vpisno številko 63030130,

sem avtor diplomskega dela z naslovom:

Optimizacija poizvedb v sistemu ElasticSearch.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Marka Bajca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) in ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 25. 8. 2016 Podpis avtorja:

Zahvala

Zahvaljujem se fakulteti za podano uporabno znanje.
Zahvaljujem se staršema za vso podporo med časom študija.

Diplomo posvečam svoji utrujeni vesti.

Kazalo

Seznam uporabljenih kratic in simbolov

IPO	Iskalniki za podjetja in organizacije. (ang. Entepreise Search)
ELK (stack)	Elastic Search, Logstash, Kibana.
ES	Elasticsearch
JSON	Javascript object notation
REST	Representetional state transfer
HTTP	Hype text transfer protocol
ID	Identifikator
JVM	Java virtual machine
VP	vrednosti polj (eng. field data)
DV	dokumentne vrednosti (eng. doc values)

Povzetek

V diplomskem delu predstavim bazo, njeno zgodovino nastanka in kam se umešča s stališča primerov uporabe na trgu programske opreme. Naredim kratek pregled primerov uporabe iz realnega sveta in na kratko raziščem trende njene popularnosti v primerjavi s sorodnimi produkti ter trgom kot celoto. Na podlagi lastnih izkušenj, literature, uradne dokumentacije in izkušenj drugih uporabnikov obravnavam primere, ki so povzročali problematično delovanje baze. Za vsakega od primerov obravnavam možnost in smiselnost rešitve problema z avtomatsko optimizacijo poizvedb. Primer obravnavam tudi zgodovinsko, saj je Elasticsearch v zadnjem letu z novimi verzijami bistveno spreminjal. Ugotavljam, da avtomatizacija poizvedb ni smiselna, saj so razvijalci pri Elastcisearchu večino primerov rešili z arhitekturnimi spremembami, z interno optimizacijo in s spremembo poizvedovalnega jezika, ki uporabniku odvzame dvoumnost pri izražanju poizvedb. Ugotavljam, da so najbolj pomembna lastnost dobro delujoče gruče, primerna velikost črepinje, ki pa je ni mogoče enostavno spreminjati. Za to je potrebno eksperimentalno načrtovanje zmogljivost, ki ga tudi opišem. Poleg optimalne velikosti črepinje obstaja nekaj slabih praks, ki jih v delu tudi opišem, z njimi lahko sesujemo gručo in zanje je pomembno, da jih uporabnik baze pozna.

Ključne besede:

Elasticsearch, query, optimization, skalabilnost

Abstract

In the graduation thesis, I present database, its history of origin, and where it is placed from the perspective of cases of use on the software market. I make a short overview of examples of use from the real world, and shortly research trends of its popularity compared to related products and market as a whole. Based on my own experience, literature, official documentation, and experience of other users, I examine the cases which caused problematic operation of the database. For each of the cases I examine the possibility and advisability of solving the problem with automatic optimisation of queries. I examine the case also historically, since Elasticsearch has in the last year significantly changed. I note that automation of queries is not advisable, since the developers in Elasticsearch solved most of the cases with architectural changes, internal optimisation, and a change of query language, which takes away from the user ambiguity in expressing the queries. I establish that the most important feature of well-functioning cluster is a proper size of shards, which cannot be easily changed. For that, an experimental planning of activities is necessary, which I also describe. In addition to optimum size of shard, there are some bad practices, which I also describe in the thesis; with them we can collapse cluster, and it is important that they are known by the user.

Key words:

Elasticsearch, query, optimization, scalability

Poglavje 1

Uvod

Elasticsearch je novejša in vedno bolj popularna podatkovna baza NoSql. Uporablja se za indeksiranje velikih količin podatkov za namene analitike in polnotekstovnega iskanja. Njena hitro rastoča priljubljenost in nezrelost uporabnikom povzročata veliko težav. Med takšnimi uporabniki sem bil tudi sam. Bazo sem dnevno uporabljal tri leta in naletel na nemalo težav. V tem delu zbiram najbolj pogoste težave in obravnavam možnosti optimizacije poizvedb, s katerimi se jim lahko izognemo.

Poglavje 2

Elasticsearch

Elasticsearch (v nad. ES) je najbolj popularen (drugi je Apache Solr) odprtokodni iskalnik, ki temelji na osnovnejšem iskalniku Apache Lucene [11] (tudi Solr temelji na Lucene). Napisan je v jeziku Java in dostopen pod licenco Apache Licence 2.0 [10]. Prvo verzijo je izdal Shay Banon leta 2010. Danes je eden izmed več odprtokodnih produktov pod okriljem podjetja Elastic.

2.1 Zgodovina in trenutno stanje

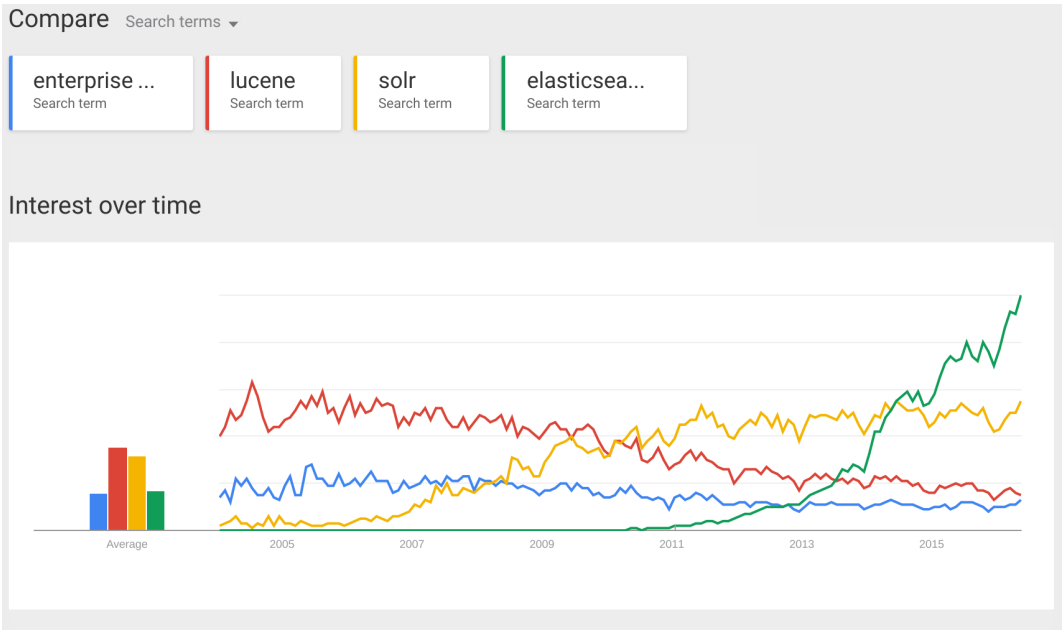
V času pisanja tega dela je produkcijska verzija baze 2.3. Na voljo je že naslednja še neizdana verzija 5.0, ki v veliki meri vključuje spremembe, ki so posledica performančnih težav prejšnjih verzij. Oštevilčenje verziji 3.0 in 4.0 so preskočili zaradi poenotenja z verzijami sorodnih produktov. S samo bazo izdajajo še komplementarne verzije drugih orodij z naslednjimi nameni uporabe: raziskovanje in vizualizacija podatkov, avtentikacija in avtorizacija dostopa (od posameznega dokumenta do celotne baze), enkripcija podatkov in komunikacijskih kanalov, črpanje podatkov iz drugih sistemov in njihovo predprocesiranje, orodja za administracijo in nadzor delovanja. Ti komplementarni produkti so od verzije 5.0 združeni pod imenom X-Pack in bodo izhajali kot dodaten vtičnik k vsaki pripadajoči verziji.

2.2 Iskalniki za podjetja

ES se je primarno uveljavljal kot skalabilno orodje za polntotekstovno iskanje, a postopoma se s pripadajočimi razširitvenimi uveljavlja na trgu iskalnikov za podjetja (eng. enterprise search). Trg se je v desetih letih zelo spreminjal.

Številni zasebni ponudniki so izginili, te pa nadomeščajo rešitve, ki temeljijo na odprtokodni knjižnici Apache Lucene. Zelo razširjena je odprtokodna rešitev Apache Solr. Najnovejša vzpenjajoča se rešitev, ki obravnavam v tem delu, je Elasticsearch. [14].

Za ponazoritev prilagam graf(slika 2.1), izsekan iz orodja Google Trends [12]. Predstavlja medsebojno relativno popularnost iskanih besed: Enterprise Seach, Lucene, Solr in Elasticsearch. Podatki so od leta 2004 do leta 2016 z območja ZDA.

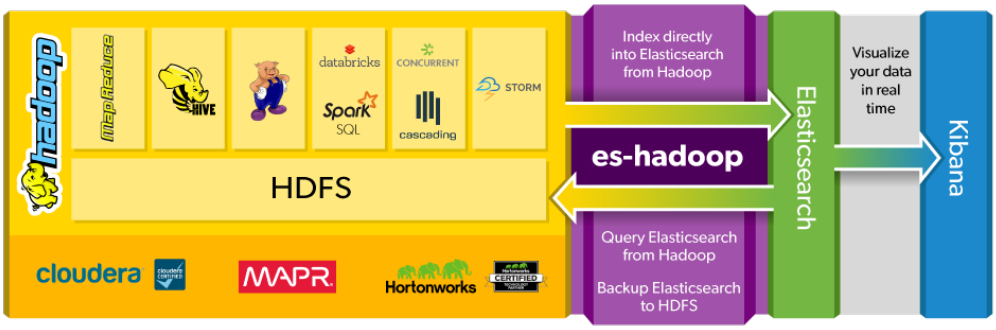


Slika 2.1: Trendi iskanja pojmov: Enterprise Seach, Lucene, Solr in Elasticsearch od leta 2014 do leta 2016 na območju ZDA.

Z grafa je razvidno, da iskanost pojma "Enterprise Search" upada. Upada tudi iskanost besede "Lucene". Beseda "Solr" je dosegla določen plato, medtem ko je "Elasticsearch" v vzponu.

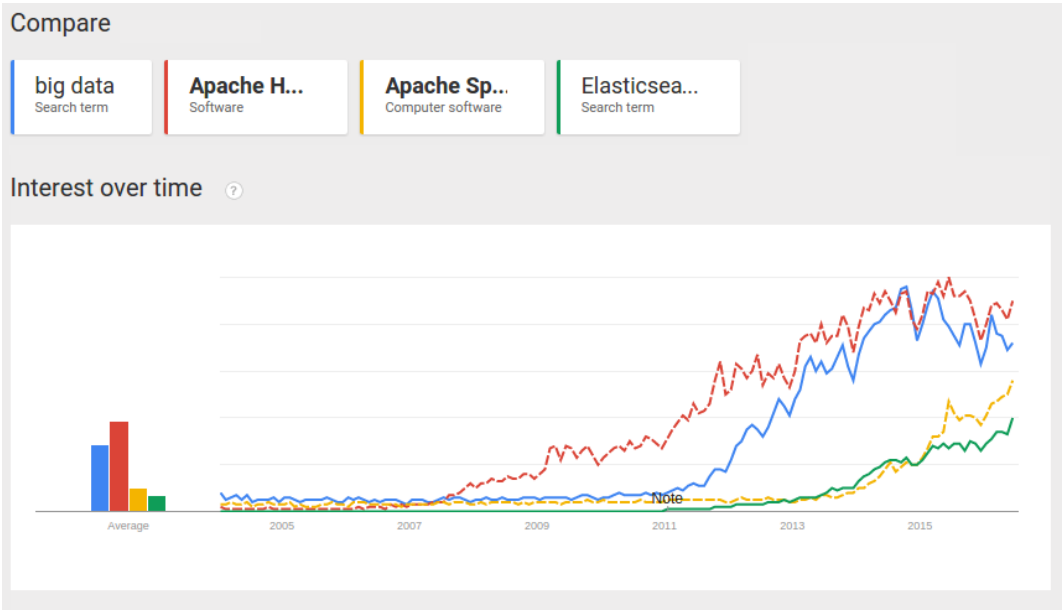
2.3 Procesiranje velikih količin podatkov (eng. Big Data)

Poleg polnotekstovnega iskanja po podatkih je primaren namen ES-ja analitika v realnem času. Ta primer uporabe bazo umešča na novejši trg programskih rešitev, imenovan "Big Data". Rezultati poizvedb se uporabljajo za vizualizacije, namenjene človeškim uporabnikom ali programom. Razširjena odprtokodna orodja za procesiranje velikih količin podatkov temeljijo na sistemu Hadoop ali komunicirajo z njim in ES ni izjema (slika 2.2) [9].

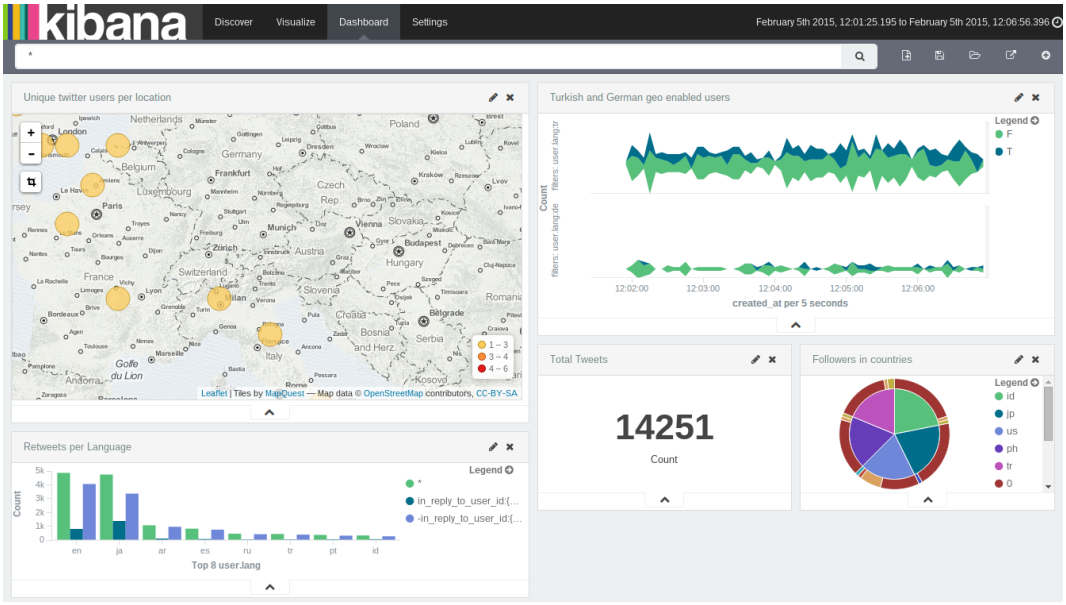


Slika 2.2: Povezljivost Elasticsearch s sistemom Hadoop

Za ponazoritev korelacije med Elasticsearcom in Hadoopom prilagam graf(slika 2.3), izsekan iz orodja Google Trends [12]. Podatki so od leta 2004 do leta 2016 z območja ZDA.



Slika 2.3: Relativna popularnost iskanih besed: Big data, Apache hadoop, Apache spark in Elasticsearch.



Slika 2.4: Primer vizualizacij v aplikaciji Kibana.

2.4 Realni primeri uporabe

Bistvena lastnost ES-ja je v tem, da omogoča skaliranje. Realni premieri uporabe so kazalniki te lastnosti.

2.4.1 Wordpress.com

Wordpress je najbolj razširjen odprtokododen CMS, napisan v PHP-ju. Fundacija uporabnikom omogoča pisanje lastnih blogov, gostovanjih na njihovih strežnikih. V času pisanja uporabljajo pet ES-gruč. [1] Največjo in najstarejšo uporabljajo za iskanje sorodnih vsebin - uporabnik bere določen članek, sistem mu ponudi druge članke s sorodno tematiko. To gručo poganja 42 podatkovnih vozlišč. Razporejena so po treh podatkovnih centrih. Tipično vozlišče ima 96 GB pomnilnika, od katerega je 31 GB namenjenega javanskemu ES-procesu. Za hrambo uporabljajo SSD-diske s kapaciteto od 1 TB do 3 TB na vozlišče. Celotno gručo sestavlja 1925 črepinj in vsebuje 11 milijard dokumentov. Tipičen indeks vsebuje 175 črepinj in vsebuje 10 milijonov blogov. Pri indeksiranju blogov uporabljajo usmerjanje (eng. routing), ki se izkorišča pri poizvedbah. Posledično večina poizvedb uporablja samo eno gručo. Dnevno dodajo ali posodobijo 12 milijonov dokumentov (z rekordom 940 milijonov v enem dnevu) in postrežejo 60 milijonov poizvedb. Odzivni časi poizvedb: mediana 44 ms; 95-ti percentil: 190 ms; 99-ti percentil: 650 ms.

2.4.2 Drugi

Na www.elastic.co/use-cases je predstavljana vrsta velikih podjetij in organizacij, ki uporabljajo ES. Med primeri uporabe najdemo: dodajanje polnotekstovnega iskanje različnim obstoječim sistemom, zbiranje dnevniških datotek, iskanje podobnih dokumentov trenutnemu, zbiranje podatkov o infrastrukturi in obveščanje o nepričakovanih dogodkih, analizo prometnih podatkov za identifikacijo napadov, ugotavljanje trendov iz aktivnosti uporabnikov.

Poglavje 3

Osnovni pojmi

3.1 Logična in fizična organizacija podatkov

Za razumevanje organiziranosti podatkov je potrebno razumevanje terminologije.

Gruča (eng. Cluster) Ena instanca ES-baze, do katere lahko dostopamo. Sestavljena je iz najmanj enega ali več vozlišč. Komuniciramo lahko z vsakim vozliščem preko vmesnika (HTTP, REST, JSON).

Vozlišče (eng. Node) En ES-proces, ki običajno živi na enem (virtualnem) računalniku. Vsako vozlišče je ekvivalentno in sposobno usmerjati poizvedbe po ostalih vozliščih, sprejeti delne rezultate in jih združiti v odgovor.

Index (eng. Indeks) Je logična množica dokumentov, po kateri lahko iščemo. En indeks je razkosan na eno ali več disjunktnih črepinj (shards), ki so porazdeljene po vozliščih (nodes). En indeks lahko hrani dokumente različnih tipov (type).

Tip (eng. Type) Dokument je vedno del enega indeksa in v kontekstu indeksa ima določen tip. Dokumenti istega tipa imajo definirano shemo oz. preslikavo, ki jo v ES-terminologiji imenujemo "mapping". Shema je praviloma definirana pred vnosom dokumentov, lahko pa je tudi fleksibilna in se gradi sproti na podlagi vhodnih podatkov ter konfiguracije.

Črepinja (eng. Shard) Je najmanjša logična enota, ki se lahko premakne z enega vozlišča na drugo. Stvarno pa je to ena instanca Lucene, ki živi na enem vozlišču. Logično pripada enemu indeksu in hrani dokumente

različnih tipov (type), pripadajočih temu indeksu. Črepinja je lahko primarna (primary) ali replika (replica). Poslani podatki se najprej vpišejo v primarno črepinjo, nato pa podvajajo na pripadajočo repliko.

Segment (eng. Segment) Je del indeksa Lucene. Hrani originalne dokumente, invertiran indeks, pogostost pojavljanja nizov za vsak dokument in pogostost nizov v celotnem segmentu. Segment, zapisan na disk, se ne spremeni. Dodajanje, spreminjanje in brisanje podatkov ustvarijo nove segmente, ki so v rednih intervalih združeni v nove segmente. Posledično segmenti hranijo tudi seznam zbrisanih dokumentov in nove verzije spremenjenih dokumentov.

Dokument (eng. Document) Je najmanjša podatkovna enota, ki jo lahko indeksiramo in po kateri lahko poizvedujemo. Dokumenti so predstavljeni v JSON-obliki. Vedno pripadajo enemu tipu in enemu indeksu. Dokument sestavljajo polja (eng. fields), v katerih so prisotni podatki.

Polje (eng. Field) Je del dokumenta in hrani določen podatek. Tip podatka, ki ga polje hrani, je definiran v shemi, imenovani "mapping". Vsak podatkovni tip ima definirano shemo, ki določa, katera polja lahko dokument vsebuje, in vrsto lastnosti za vsako polje posebej.

Invertiran indeks (eng. Inverted index) Je preslikava nizov, ki v sistem vstopajo kot vrednosti polj dokumentov. Za vsak niz obstaja seznam ID-jev dokumentov. Te indekse upravlja Lucene.

3.1.1 Poizvedbe

Z ES komuniciramo preko HTTP-vmesnika s pošiljanjem(3.1) in sprejemanjem (3.2) JSON-dokumentov. ES-dokumente točkuje glede na relevanco, razvrsti in vrne omejeno število najbolje točkovanih. Rezultate agregacij dobimo v istem odgovoru kot dokumente. Zahtevamo lahko samo dokumente, samo agregacije ali oboje skupaj. Poizvedbe lahko razumemo kot sintaktično drevo, kjer končna vozlišča iščejo določeno vrednost v določenem polju, in vmesna vozlišča, ki logično združujejo končna vozlišča. Predikate lahko izrazimo v dveh kontekstih, in sicer v poizvedbenem kontekstu (query) ali filtrirnem kontekstu (filter).

Izvorna koda 3.1: Poizvedba vsebuje pogoj v poizvedbenem kontekstu, filtrirnem kontekstsu in agregacijo

```
{
    "query": {
      "bool": {
          "must": [
            { "match": { "description":   "cotton"  }
               },
          ],
      },
      "filter": [
        { "term":  { "category": "clothes" }},
      ]
    },
    "aggs" : {
        "sub_categories" : {
            "terms" : { "field" : "category" }
        }
    },
    "size": 10,
    "offset": 10
}
```

Izvorna koda 3.2: Odgovor vsebuje en dokumentom in rezultat agregacije

```
{
  "took" : 26,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1000,
    "max_score" : 3.1,
    "hits" : [ {
      "_index" : "products",
      "_type" : "product",
      "_id" : "1",
      "_score" : 3.1, "_source" : {...}
    } ]
  },
    "aggregations" : {
        "sub_categories" : {
            "doc_count_error_upper_bound": 0,
            "sum_other_doc_count": 0,
            "buckets" : [
                {
                    "key" : "shorts",
                    "doc_count" : 12
                },
                {
                    "key" : "shirts",
                    "doc_count" : 15
                },
            ]
        }
    }
}
```

Distribucija poizvedbe

Klient pošlje poizvedbo do enega od vozlišč. To v kontekstu poizvedbe imenujemo koordinirno vozlišče. V poizvedovalni fazi [4] koordinirno vozlišče po sistemu round-robin izbere črepinje, na katere posreduje poizvedbo. Ta vedno vsebuje tudi število želenih dokumentov (N) in odmik (O). Če vrednosti v poizvedbi niso navedene, se uporabi privzeto število N = 10 in odmik O = 0. Na vsaki od črepinj poizvedba rezultira v urejen seznam parov ID-jev in pripadajočih sortirnih vrednosti velikost O+N. Ti seznami so vrnjeni na koordinirno vozlišče. Sledi sprejemna faza. Koordinirno vozlišče združi prejete sezname v urejen seznam ID-jev ter zavrže tiste ID-je, ki ne ustrezajo parametrom odmika in želenega števila dokumentov. Za ostale ID-je od istih črepinj, od koder so prišli, in zahteva ter prejme dokumente in oblikuje odgovor, ki ga vrne klientu. Torej, če na primer zahtevo drugih (O = 10) najboljših 10 (N = 10) dokumentov in je indeks, kjer poizvedujemo, razdeljen na 5 črepinj, bo vsaka izmed črepinj vrnila seznam 20 ID-jev ter sortirnih atributov, torej skupno 100 (5*20). Prvih 10 in zadnjih 80 bo zavrženih, za ostalih 10 dobimo urejen seznam najboljših dokumentov.

Poglavje 4

Optimizacije

V tem poglavju obravnavam različne primere optimizacije poizvedb. Primere sem zbral na podlagi lastnih izkušenj in pogostih priporočil optimizacij iz literature: uradna dokumentacija, prispevki uporabnikov, prispevki podjetij, ki prodajajo gostovanje, in vzdrževanje ES-gruč.

4.1 Izvajanje meritev

Meritve poizvedb sem izvajal z orodjem esrally [7]. To je odprtokodno orodje, s katerim razvijalci ES-a avtomatizirano izvajao meritve delovanja zadnje objavljane kode. Rezultati meritev so javni [8]. Orodje omogoča namestitev gruče, indeksiranje podatkov, izvajanje poizvedb in hrambo rezultatov. Sam sem postopek orodja modificiral tako, da sem določeno gručo ohranil in večkrat uporabil. Orodje pride z nekaj pripravljenimi primeri, za katere ima na voljo tudi podatke. Te avtomatsko sname s strežnika in indeksira lokalno. Meritve sem izvajal na različno velikih črepinjah, na različnem številu črepinj z namenom, da ugotovim, kako se optimizacija obnaša z rastjo podatkov v smislu posamezne črepinje ali dodajanja črepinj. V nekaterih primerih sem izoliral vpliv obravnavane poizvedbe tako, da sem upošteval njen relativni prispevek k celotnemu času odgovora. Za ta namen sem meril tudi čas strežbe za tako imenovano poizvedbo "match_all". Ta čas vključenje pot do baze in nazaj, prenos podatkov, prevzem dokumentov (privzeto 10) ter gradnjo odgovora. To so operacije, ki se zgodijo vedno, ne glede na specifiko opazovane poizvedbe. Vsi eksperimenti so najprej izvedli 2000 poizvedb, ki se ne upoštevajo v meritvah, nato pa 6000 takšnih, ki se upoštevajo. Vsak eksperiment sem izvedel petkrat. Od izmerjenih vrednosti sem zavrgel najboljšo in najslabšo, iz ostalih treh pa izračunal povprečje.

4.1.1 Zbirka podatkov PMC

PMC-zbirka podatkov prihaja skupaj z orodjem esrally. Vsebuje petsto tisoč medicinskih člankov. Namenjena je testiranju poizvedb nad dokumenti v angleščini. Bazo sem indeksiral ne pet različnih načinov v odvisnosti od celotne količine podatkov in njihove razdelitve po črepinjah. Za vsako od črepinje sem izmeril čas minimalne4.1 poizvedbe in čas poizvedbe brez pogojev, ki vrne deset dokumentov. Baza podatkov je na disku velika 22 GB.

ime gruče	velikost indeksa	število črepinj	velikost črepinje	match_all[ms]
100k1s	100.000	1	100	8,3
200k1s	200.000	1	200	9,2
200k2s	200.000	2	100	8,7
500k1s	500.000	1	500	12,4
500k5s	500.000	5	100	11,4

Tabela 4.1: Osnovne lastnosti petih gruč medinskih člankov.

4.2 Filtrirni in poizvedovalni kontekst

Razlika med izražanjem predikatov v različnih kontekstih (poizvedovalni, filtrirni) je v tem, da se pri filtriranju ne izračunavajo točke. Dodatno se filtri s predpomnjenjem izognejo določenemu delu operacij4.3. Pri poizvedovalnem kontekstu se pri iskanju dokumentov po predikatu (polje, vrednost) dostopa do invertiranega indeksa. Ta za iskani niz/vrednost vsebuje seznam ID-jev in za vsakega od ID-jev še frekvence pojavljanja vrednosti v danem polju. Te frekvence se poleg globalnih frekvenc uporabljajo za izračun točk (relevance) za vsakega od ID-jev. Nasprotno se pri filtrih uporabijo samo ID-ji, točkovanje se ne izvaja. Splošna praksa je, da poizvedovalni kontekst uporabljamo, kadar delamo polnotekstovno iskanje ali želimo, da neki pogoj prispeva k točkovanju. V vseh ostalih primerih uporabljamo filtre. [6] Razliko med poizvedovalnim in filtrirnim kontekstom sem testiral s poizvedbami na bazi PMC4.1.1. Prvi eksperiment izvaja iskanje po vsebini članka z besedico “the”. Beseda je najbolj splošna možna beseda, z namenom da mora ES obdelati kar se da veliko zalogo vrednosti, z upanjem, da bo prišla razlika med filtri in poizvedbami do izraza. Filter in poizvedba bosta obdelala enako število dokumentov.

Izvorna koda 4.1: Poizvedba Q, ki povzroči točkovanje kar se da velike množice dokumentov.

```
{
  "query": {
    "bool": {
      "must": [
        {"match": {"body": "the"}}
      ]
    }
  }
}
```

Izvorna koda 4.2: Poizvedba F, ki povzroči filtracijo kar se da velike množice dokumentov.

```
{
  "query": {
    "bool": {
      "filter": [
        {"match":{"body":"the"}}
      ]
    }
  }
}
```

Iz rezultatov meritev (tabela 4.2) je razvidno, da se časi strežbe zahtev pričakovano večajo z velikostjo črepinj, manjšajo pa z razkosanjem podatkov na manjše črepinje. Razvidno je tudi, da je zaradi režije med več črepinjami potreben določen delež časa, a je ta relativno majhen. Razlika med poizvedbo in filtrom je vedno prisotna, a raste z velikostjo črepinje.

ime gruče	Q[ms]	F[ms]	Q/F	QB[ms]	FB[ms]	QB/FB
100k1s	14,58	11,72	1,24	6,28	3,42	1,84
200k1s	17,03	13,37	1,27	7,83	4,17	1,88
200k2s	15,81	12,19	1,30	7,11	3,49	2,04
500k1s	28,73	20,06	1,43	16,33	7,66	2,13
500k5s	17,58	15,61	1,13	6,18	4,21	1,47

Tabela 4.2: Primerjava poizvedb Q in F.

V praktični uporabi ES-ja lahko pričakujemo, da bo v poizvedbi nastopalo več pogojev hkrati. Za ta namen sem izmeril tudi razliko med poizvedbo in filtrom, ko nastopajo trije pogoji. Rezultati meritve so predstavljani v tabeli 4.3. V tem primeru se časi strežbe pričakovano podaljšajo, vendar se razmerje med poizvedbo in filtrom bistveno poveča šele pri največji črepinji. Iz meritev sklepam, da so filtri vedno hitrejši od poizvedb, razlika pa se povečuje z rastjo črepinje, veliko manj pa z rastjo števila črepinj.

ime gruče	Q[ms]	F[ms]	Q/F	QB[ms]	FB[ms]	QB/FB
100k1s	19,48	15,03	1,30	11,18	6,73	1,66
200k1s	25,85	18,18	1,35	16,65	9,98	1,67
200k2s	19,83	14,96	1,33	11,13	6.26	1,78
500k1s	67,11	29,90	2,24	54,71	17,5	3,13
500k5s	26,21	20,49	1,28	14,81	9,09	1,63

Tabela 4.3: Primerjava poizvedb QQQ in FFF.

4.3 Predpomnjenje filtrov

ES pri izvajanju filtracije podatkov (filtrirni kontekst, brez točkovanja) lahko uporablja predpomnjenje. [2]. V starejših verzijah je lahko programer v poizvedbo (filter) dodal direktivo o tem, ali naj se filter predpomni ali ne. Posledično so bile te direktive predmet razprave za optimizacijo poizvedb. Z zadnjo verzijo se je to bistveno spremenilo. Predpomenjenje filtrov se izvaja v kontekstu segmentov. Na podlagi filtra (par polje, vrednost) se iz invertiranega indeksa izvleče seznam ID-jev (dokumentov), ki temu pogoju ustrezajo. V kontekstu enega filtra se zgradi bitno polje, ki vsebuje vse ID-je dokumentov, prisotnih v segmentu. ID-ji, ki ustrezajo filtru, so označeni z 1, ostali z 0. Bitno polje je implementirano kot Roaring Bitmap [13]. Implementacija izkorišča gosto ali redko distribucijo vrednosti za učinkovito porabo pomnilnika, s tem da omogoča učinkovite spremembe. Ta bitna polja so predmet predpomnjenja in so neodvisna od posamezne poizvedbe. Če dve različni poizvedbi uporabljata enak filter (polje, vrednost), se za obe uporabi isto bitno polje. Izgradnja bitnega polja ni vedno smiselna, pri manjših segmentih s kratko življenjsko dobo je lahko tudi škodljiva. Za ta namen ES ne gradi bitnih polj v segmentih, ki predstavljajo manj kot 3% celotnega indeksa ali vsebujejo manj kot 10000 dokumentov. Poleg tega se bitna polja ne predpomnijo takoj, ampak šele ko

je frekvenca pojavljanja filtra dovolj visoka. ES interno šteje vsako uporabo vsakega filtra in samo dovolj frekventne predpomni. Kadar zmanjka za to namenjenega prostora, najmanj uporabljene zavrže. Bitna polja so posodobljena v realnem času. S tem ko se spreminjajo in dodajajo dokumenti v segment, se predpomnjenih bitnih polje ne zavrže, ampak posodobi. Iz opisanega načina delovanja je razvidno, da so razvijalci ES-ja implementirali mehanizme, ki naredijo razpravo o predpomenju pri novejših verzijah brezpredmetno. Iz poizvedb so tudi umaknili možnost dodajanja direktiv za predpomnjenje filtrov.

4.3.1 Optimizacija časovnih izrazov

Predpomnjenje filtrov se izvaja avtomatsko. Vendar lahko z izražanjem interpretiranih predikatov po nepotrebnem generiramo fitre, ki predpomnjenja ne koristijo. Filter Rd 4.3 generira vrednost (čas izražen v milisekundah), ki se vsaj dvanajst ur ne spreminja. Filter Rms 4.4 generira vrednost, ki se spreminja vsako milisekundo. Posledično je verjetnost predpomonjenja in ponovne uporabe filtra Rms skoraj nična. Razliko med filtroma sem tistiral na gruči 500k5s. Izvedel sem pet zaporednih meritev iz katerih je so razvidni konstanto boljši časi strežbe filtra Rd v primerjavi z filtrom Rms. Rezultati so predstavljeni v tabeli 4.4.

Izvorna koda 4.3: Filter Rd, zaokrožen na en dan natančno.

```
{
  "range": {
    "timestamp":{
      "gte": "now-10y/d"
    }
  }
}
```

Izvorna koda 4.4: Filter Rms, brez zaokrožitve.

```
{
  "range": {
    "timestamp":{
      "gte": "now-10y"
    }
  }
}
```

meritev	Rd[ms]	Rms[ms]	Rd/Rms
1	20.1501	23.2251	0.87
2	18.4517	20.0851	0.92
3	17.9415	20.7954	0.86
4	18.6212	20.8545	0.89
5	17.8626	19.565	0.91

Tabela 4.4: Primerjava filtrov Rd in Rms.

4.4 Vrstni red predikatov

Poizvedbe običajno vsebujejo več filtrov, povezanih z različnimi operatorji. V filtrirnem kontekstu ES zgradi ali dobi iz predpomnilnika bitna polja za vsak filter posebej ter iterira čeznje in sestavi množico ID-jev (dokumentov), ki ustrezajo vsem filtrom. Pojavi se vprašanje, kakšen je optimalen vrstni red iteracije in ali ima pisec poizvedbe kakršenkoli vpliv na hitrost delovanja, s tem ko v poizvedbi zamenja vrstni red filtrov. Izkaže se, da je to odvisno od verzije ES-ja. V starih verzijah (manjše od 2.0) je bila dobra praksa dajanje prednosti filtrom, ki zavrnejo največ dokumentov. V novejših verzijah pa se ES sam na podlagi hevristik odloča interno. Na splošno pa najprej iterira čez bitna polja z najmanj enicami - torej najmanjšo množico dokumentov. [5] To pomeni, da se dobra praksa zdaj izvaja avtomatično in je s stališča programerja vrstni red nepomemben. Na bazi PMC sem izvedel 4 poizvedbe na gručah 500k1s in 500k5s: F-filter, ki zajema skoraj vse podatke (99.98%); f-filter, ki zajame podmnožico podatkov(31.10%; Ff(4.5)- filter presek množic, kjer je večja množica F na prvem mestu; fF(4.6)- filter presek, kjer je manjša množica f na prvem mestu. Pričakoval sem, da bo razmerje med filtri F/f relativno veliko, saj f predstavlja približno za tretjino manjšo množico dokumentov. Razmerje med Ff/fF pa blizu 1, saj po pričakovanju iz dokumentacij, odločitev o vrstnem redu sprejema ES. Iz meritev 4.5 je za gručo 500k1s razmerje med Ff/fF enako 1,03, kar favorizira neoptimalen vrstni red. Za gručo 500k5s, kjer so podatki razporejeni na pet črepinj, je razmerje enako 1. Enake predikate(4.7,4.7) sem uporabil tudi v poizvedenem kontekstu (tabela 4.6). Opazni so višji odzivni časi. Razloge za to lahko iščemo v poglavju 4.2.

Izvorna koda 4.5: Filter Ff z bolj specifičnim predikatom na drugem mestu.

```
{
  "query": {
    "bool": {
      "filter": [
        {"match":{"body":"the"}},
        {"match":{"body":"cancer"}}
      ]
    }
  }
}
```

Izvorna koda 4.6: Filter fF z bolj specifičnim predikatom na prvem mestu.

```
{
  "query": {
    "bool": {
      "filter": [
        {"match":{"body":"cancer"}},
        {"match":{"body":"the"}}
      ]
    }
  }
}
```

ime gruče	F[ms]	f[ms]	Ff[ms]	fF[ms]	fF/Ff
500k1s	24,77	15,54	17,75	18,33	1,03
500k5s	16,58	14,10	15,13	15,11	1,00

Tabela 4.5: Primerjava vrstnega reda filtrirnih predikatov. Manjši kot je vpliv vrstnega reda, bliže vrednosti 1 so vrednosti v prvem stolpcu.

Izvorna koda 4.7: Filter Qq z bolj specifičnim predikatom na drugem mestu.

```
{
  "query": {
    "bool": {
      "must": [
        {"match":{"body":"the"}},
        {"match":{"body":"cancer"}}
      ]
    }
  }
}
```

Izvorna koda 4.8: Filter qQ z bolj specifičnim predikatom na prvem mestu.

```
{
  "query": {
    "bool": {
      "must": [
        {"match":{"body":"cancer"}},
        {"match":{"body":"the"}}
      ]
    }
  }
}
```

ime gruče	Q[ms]	q[ms]	Qq[ms]	qQ[ms]	qQ/Qq
500k1s	33,1	17,44	25,46	25,78	1,01
500k5s	18,47	12,51	15,55	15,82	0,98

Tabela 4.6: Primerjava vrstnega reda poizvedbenih predikatov. Manjši kot je vpliv vrstnega reda, bliže vrednosti 1 so vrednosti v prvem stolpcu.

4.5 Problemi s pomnilnikom pri agregacijah

4.5.1 Vrednosti polj - (eng. field data)

Eden večjih problemov v preteklih verzijah ES-a je bilo hitro poslabšanja delovanja z rastjo števila unikatnih vrednosti v poljih, nad katerimi se izvajajo

agregacije ali sortiranje. Problemi so se izražali v tem, da se je JVM, ki poganja ES-proces, sesul zaradi pomanjkanja pomnilnika (Out Of Memory Error). To je lahko povzročilo delno izgubo podatkov in je zahtevalo ročno administracijo strežnika. Vzrok teh težav je bila podatkovna struktura, imenovana "vrednosti polj" (ang Field Data, v nadaljevanju VP). Strukture je preslikava para (ID, polje) v vrednosti. Ta struktura se gradi po potrebi in se hrani v JVM-kopici. Njen problem je, da dobesedno naloži vse vrednosti nekega polja v pomnilnik. Problema so se razvijalci kratkoročno lotili tako, da so omejili količino pomnilnika, ki ga ima VP na voljo, dodatno so vnesli še mehanizme, ki ocenijo porabo pomnilnika, preden se ta zasede. Če je ocenjena poraba previsoka, se poizvedbe prekinejo. S tem so preprečili sesuvanje JVM-procesa, vendar so ostale nedelujoče poizvedbe in pogosto izvajanje čiščenja JVM-pomnilnika s tako imenovanim "nabiralcem smeti" (eng. garbage collector). Ta ima tudi sila neprijetno lastnost, da povzroča situacijo, imenovano "ustavljanje sveta" (eng. stop the world), kar pomeni, da JVM za določen čas ne počne popolnoma nič in poizvedbe čakajo, da situacija mine. Ko gruča zaide v takšno stanje, je njena uporabnost zelo degradirana. Opisani problem ni imel pametne rešitve do verzije 2.0. V organizaciji wordpress.com so kot rešitev onemogočili izvajanje agregacij po poljih z visoko kardinalnostjo.

4.5.2 Dokumentne vrednosti - (eng Doc values)

Od verzije 2.0 naprej agregacije in sortiranje interno uporabljajo strukturo, imenovano dokumentne vrednosti [3] (v nad. DV). DV je nastal kot rešitve problemov z uporabo VP (4.5.1). V verziji 2.0 je treba DV v shemi za posamezna polja eksplicitno vklopiti. Privzeto se še vedno uporablja VP. Od verzije 5.0 naprej se DV uporablja avtomatsko na vseh neanaliziranih poljih. Analizirana polja so namenjena polnotekstovnemu iskanju. VP je še vedno na voljo za agregacije na analiziranih poljih, ampak je privzeto onemogočen. Za uporabo VP ni treba reindeksirati podatkov. Bistvena razlika med VP in DV je v tem, da se DV hrani na disku in se nalaga v pomnilnik pod nadzorom operacijskega sistema. To pomeni, da nima vpliva na delovanje JVM-ja in njegovo porabo pomnilnika. DV izvaja serijo optimizacij, ki zmanjšajo velikost podatkov, in izrablja predpomnjenje pomnilnika pod nadzorom operacijskega sistema. S tem so se izognili vsem problemom, povezanimi z VP 4.5.1. Ker se DV hrani na disku, lahko pričakujemo počasnejše delovanje, vendar se DV za razliko od VP gradi sproti pri indeksiranju dokumentov. To pomeni, da se pri nastopu poizvedbe DP samo naloži iz diska v pomnilnik, medtem ko mora VP prebrati vse vrednosti v polju in sestaviti v JVM-kopico. Izkazalo se je

da, DV dobro deluje in pri ES-ju načrtujejo podobno zamenjavo tudi za edini primer, ko je VP še v uporabi. DV se za posamezna polja lahko izklopi, kar prihrani prostor na disku, vendar na tem polju ni mogoče izvajati agregacij in sortiranja.

Doc_1	100
Doc_2	1000
Doc_3	1500
Doc_4	1200
Doc_5	300
Doc_6	1900
Doc_7	4200

Tabela 4.7: Primer dokumentnih vrednosti (DV) iz ES-dokumentacije.

4.5.3 Optimizacije dokumentinih vrednosti

Optimizacije povezane z DV in VP niso vezane na same poizvedbe. Te probleme rešujemo z reindeksiranjem podatkov, drugačnimi nastavitvami sheme in uporabo novejše verzije baze. Od verzije 5.0 naprej je skoraj nemogoče povzročiti te probleme, saj so pri ES-u zavgli podatkovni tip "string" in ga nadomestili z dvema novima tipoma: "text" in "keyword". Odstranili so dvoumnost o namenu uporabe podatkovih tipov in poskrbeli za privzete nastavitve, ki preprečujejo problematične situacije.

Poglavje 5

Razprava

V tem poglavju najprej razpravljam o posameznih optimizacijah in kakšen je njihov pomen za morebitno izgradnjo avtomatskega optimizatorja. V nadaljevanju pa razpravljam o drugih praksah, ki niso nujno povezane z optimizacijo poizvedb, vendar ugotavljam, da je njihovo upoštevanje bistveno za dobro delovanje gruče.

5.1 Razprava o optimizacijah

5.1.1 Filtrirni in poizvedovalni kontekst

Iz meritev ugotavljam, da je filtrirni kontekst pri dovoljšnji količini podatkov hitrejši od poizvedovalnega. Tukaj je treba le slediti priporočilom iz dokumentacije. Avtomatski optimizator bi lahko zaznal, kdaj poizvedbe uporabljajo poizvedovalni kontekst, in uporabnika opozoril, naj premisli, ali je to res potrebno. Menim, da tole prinaša izredno malo dodane vrednosti, saj bi takšno navodilo bilo uporabno, dokler si uporabnik tega ne zapomni, potem pa moteče. Dodatno sem ugotovil, da odzivni časi in sama razlika med kontekstoma naraščajo z velikostjo črepinje, pri tem da razkosanje velikih črepinj na manjše znatno zmanjša servirne čase.

5.1.2 Vrstni red predikatov

Meritve niso pokazale bistvenega vpliva pri vrstnem redu izražanja predikatov, poleg tega se v novejši verziji ES na podlagi dokumentacije ES hevristično odloča o vrstnem redu. To naredi avtomatski optimizator za ta primer neuporaben, saj lahko ES prepiše kakršnokoli spremembo, ki jo ta naredi. Dodatno

sem ugotovil, da odzivni časi in sama razlika med kontekstoma naraščajo z velikostjo črepinje, pri tem da razkosanje velikih črepinj na manjše znatno zmanjša servirne čase.

5.1.3 Predpomnjenje filtrov

Pri ES so z novimi verzijami tako zelo spremenili predpomnjenje filtrov, da novejše verzije v poizvedovalnem jeziku ne omogočajo več ročnega upravljanja s tem predpomnilnikom. S stališča avtomatskega optimizatorja to spet pomeni, da ne more ničesar prispevati. Našel sem primere, ko so pri starejših verzijah uporabniki implementirali optimizacijo predpomnjenja na način, da so filtrirne izraze kar se da poenostavili in omogočili predpomnjenje enostavnih gradnikov ter s tem povečali verjetnost njihove ponovne uporabe. Ravno to ES zdaj počne avtomamatsko, skupaj z optimizacijami, opisanimi v poglavju 4.3. Obstajajo pa poizvedbe, ki po nepotrebnem v predikatih vedno vsebujejo unikatne vrednosti. Te ne koristijo predpomnjenja in njihova optimizacija je smiselna, vendar optimizacije ni mogoče avtomatizirati, ker je odvisna od aplikacijske logike izven podatkovne baze.

5.1.4 Problemi s pomnilnikom pri agregacijah

Opisani problemi so bili zalo pogosti in problematični, vendar jih ni mogoče reševati z drugačnimi poizvedbami, temveč z uporabo novejše verzije baze in reindeksiranjem podatkov. Tudi v novejših verzijah obstajajo situacije, kjer je lahko kardinalnost polja prevelika, da bi bilo mogoče nad njimi izvajati agregacije. Tudi v tem primeru se strežni časi lahko obvladujejo z zmanjšanjem velikosti črepinje in s povečanjem števila črepinj. To pa zahteva dodatno količino pomnilnika. Z razkosanjem indeksa na več črepinj se efektivna velikost DV zaradi podvajanja podatkov po črepinjah še poveča. Posledično je treba nove črepinje namestiti na nova vozlišča, kar pomeni višanje stroškov.

5.2 Načrtovanje velikosti črepinje

Ugotavljanje velikosti črepinje je smiselno s poganjanjem testov, ki čim bolj simulirajo produkcijsko uporabo baze. Ponudniki gostovanja, specializirani za gostovanje ES-gruč, imajo predefinirane zmogljivosti za posamezna vozlišča. Na podlagi dokumentacije ponudnikov ali neposredno iz ES-dokumentacije (priporočila) ocenimo količino pomnilnika, število procesorskih jeder in količino prostora na disku. Poženemo vozlišče s podobnimi lastnostmi, kot jih

nameravamo uporabljati v produkciji. To lahko naredimo lokalno ali pri ponudniku oblačnih storitev. Na vozlišče namestimo ES in ustvarimo prazen indeks, z eno samo črepinjo brez replikacije. Zdaj začnemo s simuliranjem produkcijskega delovanja. Dodajamo dokumente, izvajamo poizvedbe, posodabljamo dokumente itd. Frekvenco operacij čim bolj približamo ocenjenim vrednostim v produkcijskem delovanju, ob tem pa nenehno dodajamo dokumente. To počnemo, dokler se črepinja ne zlomi ali postanejo odzivni časi preveliki za normalno uporabo. Ko dosežemo točko zloma, izmerimo, koliko podatkov (število dokumentov) je črepinja uspela sprejeti. To nam predstavlja zgornjo velikosti črepinje, s katero lahko ocenimo število primarnih črepinj. Vzamemo število vseh dokumentov, ki jih želimo indeksirati, število povečamo za neki varnostni faktor, na primer 20%, in delimo z izmerjeno velikostjo črepinje. S tem smo eksperimentalno definirali minimalno število črepinj za indeksiranje ciljne količine podatkov. Upoštevati moramo še replikacijo. Običajno imamo vsaj eno repliko podatkov, s katero se zavarujemo pred izpadi vozlišč in izgubo podatkov, vendar jih lahko imamo tudi več z namenom razporejanja bremena poizvedb ali vnosa podatkov. Število replik seštejmo s številom primarnih črepinj, da dobimo skupno število črepinj za obravnavani indeks. S tem smo opisali zahteve za servisiranje pisanja in branja podatkov, treba pa je razmisli še o vzdrževanju indeksa in izvajanju sprememb, ki zahtevajo ponovno indeksiranje podatkov. Reindeksiranje podatkov s stališča prostora praktično pomeni, da imamo dva indeksa z enako količino podatkov. Starega, iz katerega se podatki črpajo, in novega, v katerega se ti pišejo. Določen čas obstajata oba indeksa hkrati in šele ko obstaja dovolj zaupanja, da nov indeks lahko nadomesti starega, se poizvedbe preusmerijo na novega, starega pa izbriše. Praktična posledica je, da se v danem trenutku število črepinj podvoji, ne podvoji se pa število poizvedb in zapisov. S stroškovnega stališča je redkokdaj cena diska odločilna, posledično je smotrno, da si vnaprej zagotovimo kapacitete, ki omogočajo normalno vzdrževanje. Alternativno, ko je prostora na disku premalo, je treba kapaciteto dodati in kompleksnost vzdrževanja naraste.

$$primary\_shards = (toal\_docs\_to\_index * 1.2)/maximal\_shard\_size \tag{5.1}$$

$$total\_shards = primary\_shards * replication\_factor \tag{5.2}$$

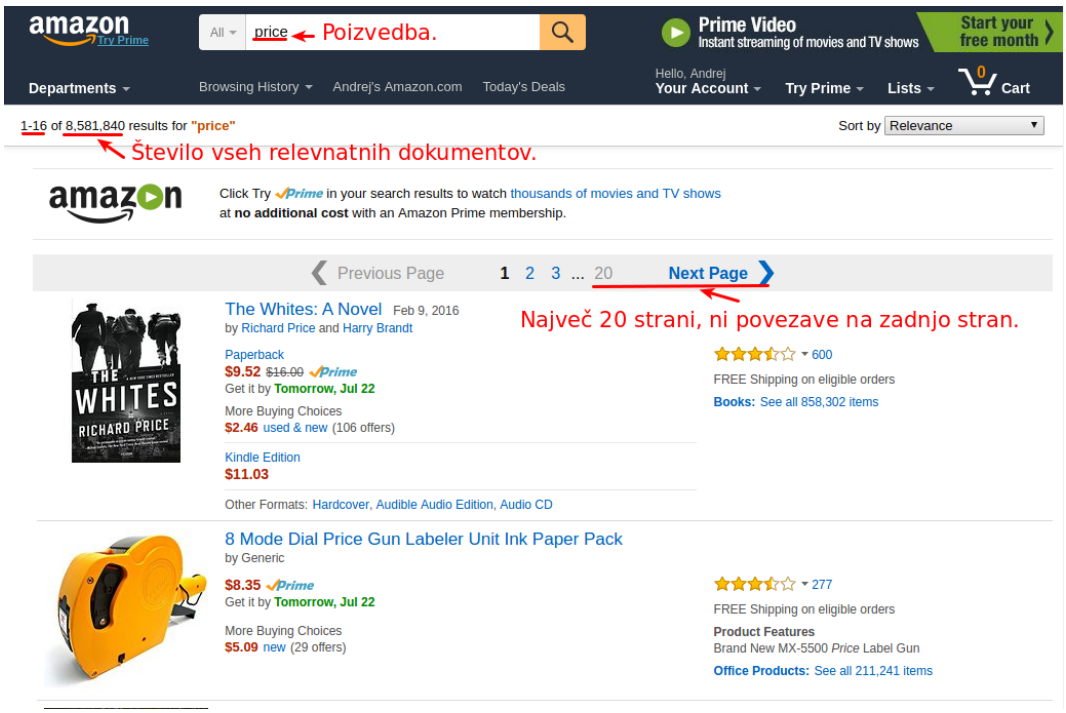
$$total\_footprint = total\_shards * shard\_size * maintanance\_factor \tag{5.3}$$

5.3 Slabe prakse

V nadaljevanju opisujem nekaj slabih praks, ki lahko povzročijo zlom gruče.

5.3.1 Globoka paginacija

Tipičen primer uporabe ES-ja je implementacija polnotekstovnega iskalnika v kontekstu spletne aplikacije, na primer spletne trgovine. Običajni uporabniški vmesnik vključuje vnosno polje, v katerega vpišemo poizvedbo in dobimo prikazan krajši seznam najbolj relevantnih izdelkov ter povezave do naslednjih nekaj izdelkov. V primerih, ko je rezultatov res veliko in uporabnik klikne na povezavo do zadnje strani, se pojavi tveganje, da bazo obremeni do njene neuporabnosti. Razloge smo nakazi že v poglavju 3.1.1. V praksi razvijalci upornikom ne omogočajo več ogledov globljih strani. Realen primer prikazuje slika 5.1 iskanja na spletni strani amazon.de. Ob iskanju niza "price" izvemo, da v njihovi bazi obstaja več kot 8 milijonov dokumentov, vendar omogočajo vpogled samo do prvih 20 strani po 15 izdelkov.



Slika 5.1: amazon.de omejuje paginacijo na 20 strani po 15 izdelkov.

Programer, ki bi naivno izdelal navigacijo, ki omogoča vpogled v zelo glo-

boke strani, lahko nenamerno povzroči zlom aplikacije. Za primer predpostavimo, da prikazujemo 10 izdelkov na stran in obiščemo stran 100.000. V ES pošlje poizvedba s parametri: odmik je 1.000.000 in število dokumentov je 10. Predpostavimo, da so podatki razdelni na 5 črepinj. To pomeni, da mora vsaka od črepinj sestaviti sortiran seznam z 1.000.010 elementi. Teh 5 seznamov se mora prenesti, da koordinirnega vozlišča, ki morajo sortirati 5.000.050 elementov in razen želenih 10 vse zavreči.

5.3.2 Dinamična shema

ES omogoča grajenje dinamične sheme na podlagi vnesenih dokumentov. Na primer: pri vnosu prvega dokumenta, ki ima v polju "nickname" vrednost "123", bo ES v shemo dodal zapis, kjer polje "nickname" definira s tipom "long". Z vnosom naslednjega dokumenta, ki ima vrednost polja nastavljeno na neki niz (npr. "andrej"), to ne deluje več, saj ne moremo v polje definirano kot število vpisovati nizov. Nepredvideni vhodni podatki lahko povzročijo, da nekaterih dokumentov ne moremo vpisati. Od programerja je odvisno, ali bodo te napake sploh zaznane. Obratna situacija nastane, ko se polje avtomatsko definira kot tekst in ne bo zavračalo števil. V praksi to onemogoča izvajanje številskih poizvedb in intervalnih agregacij. Za popravilo napake je treba reindeksirati podatke, kar je pri velikih količinah podatkov zamudno.

5.3.3 Nenadzorovana rast sheme

Dinamična shema omogoča, da za imena polj uporabimo vrednosti iz vhodnih podatkov. Hipotetični primer: indeks, imenovan "tema" (eng. topic). Za posamezno temo štejemo dogodke, ki so se v povezavi s temo zgodili vsak dan posebej. Podatke zapisujemo na način, da datum shranimo kot ime polja 5.1.

Izvorna koda 5.1: Primer dokumenta, ki za ime polja uporablja datum

```
{
  "name":"some topic",
  "activity":{
    "2016-03-03":30,
    "2016-03-04":35,
  }
}
```

To v praksi pomeni, da povzročimo situacijo, kjer je velikost sheme odvisna od vhodnih podatkov. Shema je del globalnih metapodatkov, ki so replicirani

čez vsa vozlišča. Ti podatki so vedno naloženi v Javanskem pomnilniškem prostoru. Ta se uporablja kot delovni prostor za izvajanje poizvedb in predpomnjenje. V dani situaciji je samo vršanje časa, kdaj bo ES-gruča postala nestabilna in se bo proces začel sesuvati zaradi pomanjkanja prostora. Za izhod iz situacije je treba podatke modelirati drugače 5.2, jih ponovno indeksirati v nov indeks z drugačno shemo. To je lahko problematično s stališča resursov, ker nov indeks, v katerega črpamo podatke, zahteva dodaten prostor.

Izvorna koda 5.2: Uporaba gnezdenega dokumenta

```
{
  "name":"some topic",
  "activity": [
    {
      "day":"2016-03-03",
      "count":30,
    },
    {
      "day":"2016-03-04",
      "count":35,
    },
  ]
}
```

Poglavje 6

Sklepne ugotovitve

6.1 Optimizacija poizvedb

Pregledal sem trenutno stanje ES-ja. Obravnaval sem možnost avtomatske optimizacije poizvedb na podlagi lastnih izkušenj in primerov iz literature. Prišel sem do zaključka, da izgradnja splošnega optimizatorja ni smiselna, saj so razvijalci ES-ja najbolj problematične primere z novejšimi verzijami rešili. Odločitve, ki naj bi jih sprejemal optimizator, zdaj interno sprejema ES in temu primerno prilagodi izvajanje poizvedbe.

6.2 Obvladovanje delovanja gruče

Ugotovil sem, da je najbolj pomembna lastnost dobro delujoče gruče primerna velikost črepinje, s katero obvladujemo strežne čase poizvedb. Optimalna velikost črepinje je odvisna od količine podatkov, ki jih želimo indeksirati, in poizvedb, ki jih želimo izvajati. Žal števila črepinj ne moremo dinamično spreminjati, ampak je treba podatke reindeksirati, kar zahteva čas in resurse. Zato je pomembno, da pri načrtovanju resursov črepinje načrtujemo tudi potrebe po resursih za reindeksiranje. Poleg tega je potrebno še poznavanje opisanih primerov slabih praks, ki lahko priženejo gručo do zloma.

Slike

Seznam izvorne kode

Tabele

Literatura

[1] Greg Ichneumon Brown. State of WordPress.com Elasticsearch Systems 2016, 2016. Dostopno na https://developer.wordpress.com/2016/05/03/state-of-wordpress-com-elasticsearch-systems-2016/.

[2] Zachary Tong Clinton Gormley. All About Caching, 2016. Dostopno na https://www.elastic.co/guide/en/elasticsearch/guide/master/filter-caching.html#filter-caching.

[3] Zachary Tong Clinton Gormley. Deep dive on doc values, 2016. Dostopno na https://www.elastic.co/guide/en/elasticsearch/guide/master/_deep_dive_on_doc_values.html.

[4] Zachary Tong Clinton Gormley. Elasticsearch: The Definitive Guide, 2016. Dostopno na https://www.elastic.co/guide/en/elasticsearch/guide/current/_fetch_phase.html.

[5] Zachary Tong Clinton Gormley. Internal Filter Operation, 2016. Dostopno na https://www.elastic.co/guide/en/elasticsearch/guide/master/_finding_exact_values.html#_internal_filter_operation.

[6] Zachary Tong Clinton Gormley. When to use filters, 2016. Dostopno na https://www.elastic.co/guide/en/elasticsearch/guide/current/_queries_and_filters.html#_when_to_use_which.

[7] Honza Král Nick Knize Martijn van Groningen Lyndon Swan Daniel Mitterdorfer, Michael McCandless. Macrobenchmarking framework for Elasticsearch, 2016. Dostopno na https://github.com/elastic/rally.

[8] elastic.co. Elastic Benchmarks, 2016. Dostopno na https://benchmarks.elastic.co/index.html.

[9] elastic.co. Elasticsearch Hadoop, 2016. Dostopno na https://www.elastic.co/products/hadoop.

[10] Apache Foundation. Apache Licence, Version 2.0, 2004. Dostopno na http://www.apache.org/licenses/LICENSE-2.0.html.

[11] The Apache Software Foundation. Apache Lucene Core. Dostopno na https://lucene.apache.org/core/.

[12] google. Web Search Interest: enterprise search, luecene, solr, elasticsearch - United States, 2004 - present, 2016. Dostopno na https://goo.gl/vCGxqJ.

[13] Adrien Grand. Frame of Reference and Roaring Bitmaps, 2015. Dostopno na https://www.elastic.co/blog/frame-of-reference-and-roaring-bitmaps.

[14] Miles Kehoe. One search to rule them all, 2016. Dostopno na https://www.linkedin.com/pulse/one-search-rule-them-all-miles-kehoe.